

高级语言程序设计

# 第3章

## 变量和控制结构

华中师范大学物理学院 李安邦

## 回顾：第2章学到的程序模式

程序模式1:

```
#include <iostream>
using namespace std;
int main() {
    cout << "Hello, world! " << endl;
    return 0;
}
```

程序模式2（使用数学函数）：

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
    cout << 2.0 * sin (3.14159 * 45/180) << endl;
    return 0;
}
```

已讨论机制的局限性：

$$\sum_{n=1}^{10} \sin \frac{1}{n} \quad ?$$

- 只能描述由基本数据出发的简单计算；
- 只能描述特定计算。

我们希望：

- 描述更为复杂的计算过程
- 使程序有一定**通用性**，能解决一类问题，完成对不同数据的类似计算

因此需要有更多的程序机制。

## 第 3 章 变量和控制结构

3.1 语句、复合结构和顺序程序

3.2 变量——概念、定义和使用

3.3 数据输入

3.4 关系表达式与逻辑表达式

3.5 语句与控制结构

3.6 条件语句

3.7 循环语句

3.8 程序动态除错方法（一）

## 3.1 语句、复合结构和顺序程序

- 语句（**statement**）是描述计算过程的基本单位。
- 一个语句就是由分号结束的一段字符。
- 语义：形式合法的语句表达了某种含义（程序执行时的效果），称为语句的语义。
- 语法：语句的形式必须符合语言要求。
- 语句也可以**只有一个分号**而没有其它字符，这样的语句称为**空语句**：

;

空语句执行时什么也不做，其用途就是作为填充，有时需要用它将程序的语法结构补充完整。

## 复合结构（复合语句）



- 语法：一对花括号，其中可有**0**个或多个语句。
- 语义：**顺序执行**其中的各个语句。

- **空**复合结构中没有语句，执行时立即结束。

**{ }**

- 前面简单程序的主要部分是一个复合语句：

```
int main () {  
    cout << "Hello, world! << endl;  
    return 0;  
}
```

## 第 3 章 变量和控制结构

**3.1** 语句、复合结构和顺序程序

**3.2** 变量——概念、定义和使用

**3.3** 数据输入

**3.4** 关系表达式与逻辑表达式

**3.5** 语句与控制结构

**3.6** 条件语句

**3.7** 循环语句

**3.8** 程序动态除错方法（一）

## 3.2 变量——概念、定义和使用

- 在硬件里，数据存储概念是内存单元和地址，变量是它们在高级语言里的反映。
- 变量（**variables**）：存储数据的命名对象。通过变量名可以使用存于变量中的数据。变量名是标识符。
- 程序变量与数学中的变量完全不同。一个程序变量可以看作一个数据的容器。每个变量都有一个名字，在程序中可以通过名字使用相应的变量，可以把计算中产生的数据（结果）存入变量，或者使用以前存入变量的数据。



## 3.2.1 变量的定义

- 变量有固定的类型，只能保存这个类型的值：  
整型变量（保存 **int** 值的变量），双精度变量  
（保存 **double** 值），字符变量等。

变量必须先定义（**define**）然后才能使用。



变量定义所需信息：类型和变量名。例：

```
int m;  
double x;  
char ch;
```

可以同时定义多个同类型的变量：

```
int k, n, sum, count;  
double y, z;  
char c1, c2;
```

## 变量名命名基本要求:

- 变量名是标识符，所以只能是字母（**A-Z, a-z**）和数字（**0-9**）或者下划线（**\_**）组成。而且，**第一个字符**必须是字母或者下划线开头。
- 不能使用 **C/C++** 关键字来命名变量，以免冲突。
- 在 **C/C++** 中，变量名区分大小写。

## 变量名常用规范



- 常规约定：
  - ◆ 用 **i, j, k, m, n** 或以它们开始的标识符表示整型变量（通常不用 **l**，以免与 **1** 混淆）；
  - ◆ 用 **x, y, z, a, b, c** 表示实型变量。
- 提倡采用有意义名字（**radius, sum, year...**）
- 用最短字符表示最准确的意义。
- 如果违背这些常用规范，会给自己和别人（老师、同事）带来麻烦。

大型程序中的常见命名规范:

- **骆驼式命名法**: 用单词缩写组合, 首字母大写或用下划线连接: **StudentName, StuName, InFile, OutFile, student\_name, stu\_name, in\_file, out\_file**
- **匈牙利命名法**: **变量名=属性+类型+对象描述**, 其中每一对象的名称都要求有明确含义, 可以取对象名字全称或名字的一部分。要基于容易记忆容易理解的原则。例如, 一个全局的整型变量, 用于保存学生年龄, 命名为: **g\_iStudentAge**



- 在 **ANSI C** 标准中，复合结构里的变量定义必须写在所有可执行语句之前。
- 按照 **C99** 标准和所有的 **C++** 标准，在程序中的任何复合结构里的任何位置都可以定义变量。
- 要求“先定义后使用”：变量定义应该出现在使用语句之前。
- 在一个复合结构里定义的变量可以在该复合结构的内部使用。这样的变量称为“局部变量”。

## 常见问题：变量未声明与重复声明

在程序中定义一个变量的同时，它也向编译器**声明**（**declare**）了在程序存在着这个变量。

- 变量未声明就使用，编译时会产生错误信息：

**“[Error] 'a' was not declared in this scope（'a'在这个范围内没有被声明）”。**

- 请补充写上对该变量的定义语句。

- 变量重复声明，编译时会产生错误信息：

**“[Error] redeclaration of 'int a'（重复定义'int a'）”，**

**“[Note] 'int a' previously declared here（'int a'先前在此处已被定义）”**

这时用户就应该删除对该变量的重复定义。

基本数据类型的选择:

1. 整数通常采用**int**类型
  2. 浮点数通常采用**double**类型
  3. 字符通常采用**char**类型
- 一般不要用 **unsigned** 类型

## 3.2.2 变量的使用（赋值和取值）

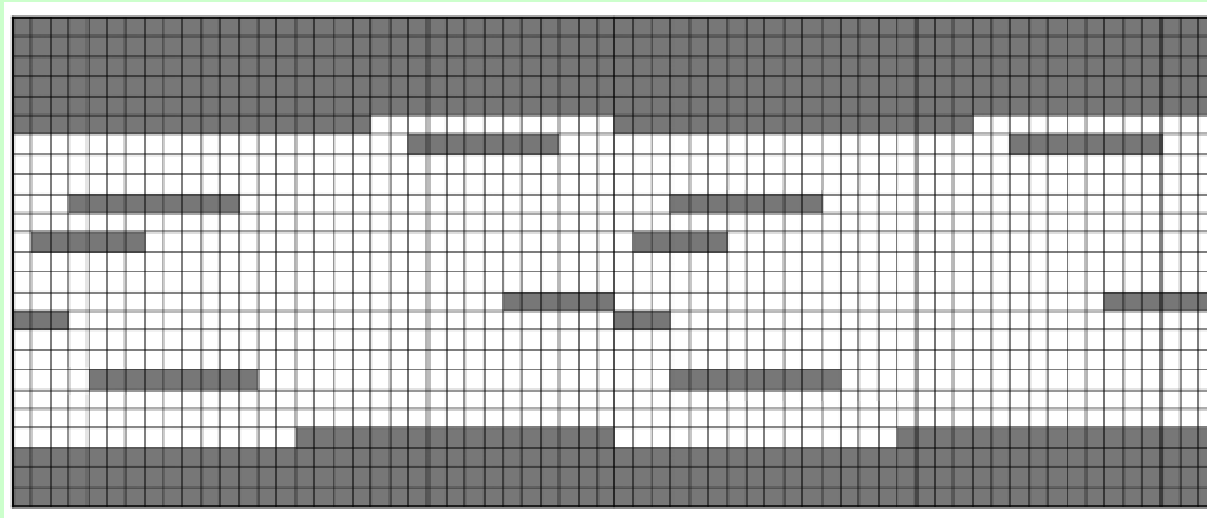
对变量的基本操作只有两个：

- 1、给变量**赋值**：将数据（值）存入变量中。
  - 2、**取值**：取得变量里当时保存的值，以便在计算过程中使用。
- 变量具有保持值的性质。赋值一次，可以多次取得相同的值。
  - 在程序中也可以对变量多次赋值（冲掉旧值）。执行中的不同时刻，一个变量里保存的值可能不同。



- 计算机内存用“内存条”一类的物理器件实现。内存包含一系列单元，常见情况是一个单元中可以存放一个字节的二进制数据。
- 内存中的每个字节有一个“地址”，软件基于内存地址访问（存入或读出）相应位置的数据。
- 内存数据访问常以计算机的字长（一个字节或多个字节）为单位。每次访问内存时，**CPU**都需要给出数据的首地址和希望访问的字节数。

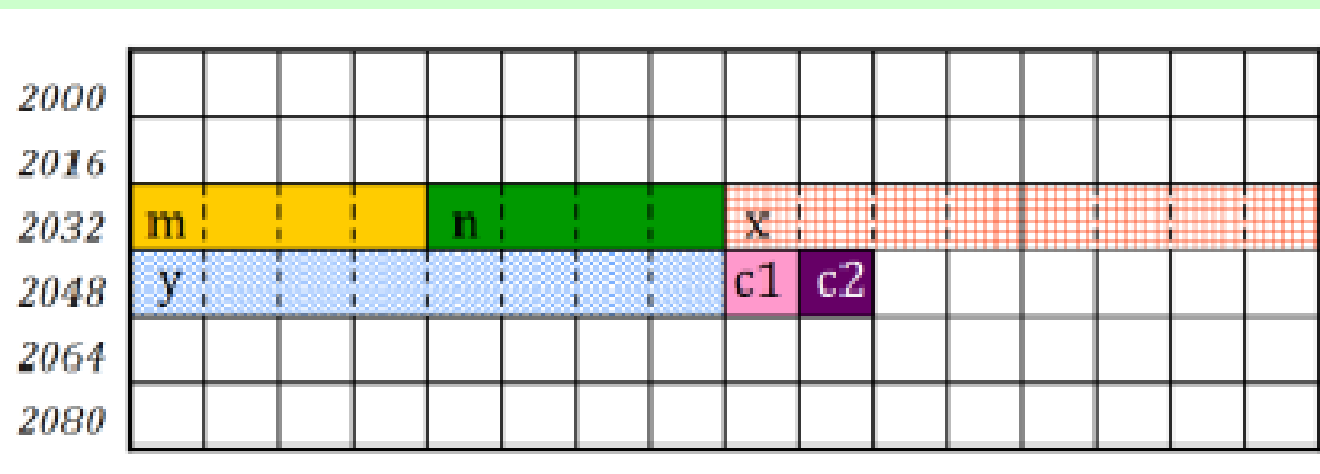
- 在计算机工作时，系统软件和应用软件中的程序和数据都需要装载到内存中，存放了程序或数据的内存称为“被占用了”。
- 一个程序在运行完毕，其所占用的内存空间就会被释放（交回计算机管理系统），空闲内存可以被其它程序重新使用。
- 在系统运行中的任一时刻，内存空间单元的占用和空闲状态可能是变化的，而且可能不是连续的。



变量功能是怎么实现的？

在程序里定义了变量，则在程序运行过程中，系统就会在合适的时候在内存中为它们分配存储位置，所分配的存储单元数由变量的类型确定。

```
int m, n;  
double x, y;  
char c1, c2;
```



变量实际上是一种可以用于存值和取值的“容器”。  
可用 **sizeof** 运算符求出变量所使用的字节数

# 一、赋值运算符和赋值表达式

赋值用赋值运算符（=，赋值号）描述。

用赋值号构造赋值表达式：

$x = 5.0$

$x = 5.0 + 3.0 * 1.5 / 20$

$y = 2 + 3 * \sin(x)$       取值

赋值目标      提供值的表达式

效果：把表达式的值赋给左边变量。

赋值运算符优先级很低（先计算表达式，再赋值）。

学习把数学公式写成 **C/C++** 表达式。例：

$$e^{\sin x + 1} \Rightarrow \exp(\sin(x) + 1)$$

$$\sqrt{\ln x + \tan x} \Rightarrow \text{sqrt}(\log(x) + \tan(x))$$


$$\log_5 \cot x \Rightarrow \frac{\ln(1 / \tan x)}{\ln 5} \Rightarrow \log(1 / \tan(x)) / \log(5)$$
$$\Rightarrow \frac{\log(1 / \tan x)}{\log 5} \Rightarrow \log_{10}(1 / \tan(x)) / \log_{10}(5)$$

二、赋值语句：赋值表达式后加分号。

是最基本的语句，完成程序里最重要的操作。

程序中一般用赋值语句描述赋值动作。

**$x = 5.0 + 3.0 * 1.5 / 20;$**

**$y = 2 + 3 * \sin(x);$**   取值

主要用途：把计算得到的中间结果存入变量，以便用于后续的计算，或者用于输出。

【例3-1】已知三角形的三条边分别是3、5、7，求这个三角形的面积。

```
#include <iostream>
#include <cmath>
using namespace std;
int main () {
    double a, b, c, s, area; //定义变量
    a = 3; //对变量赋值
    b = 5;
    c = 7;
    s = (a + b + c) / 2; //对a、b和c取值进行计算，计算结果赋值给s
    area = sqrt(s * (s - a) * (s - b) * (s - c));
    //对a、b、c和s取值进行计算，计算结果赋值给area
    cout << "Area = " << area << endl; //对area取值用于输出
    return 0;
}
```

$$s = \frac{(a + b + c)}{2}$$
$$area = \sqrt{s(s - a)(s - b)(s - c)}$$

## 赋值与类型

- 被赋值变量有类型（由变量定义确定）；
- 赋值号右边表达式的值有类型。

**规定：**若表达式值与被赋值变量类型不同，该值先转换到变量类型的值，然后赋值。

在前面程序例子里把赋值语句改写成：


**$s = (3 + 5 + 7) / 2;$**

运行时发现程序的结果不对，为什么？

（请考虑：运算在哪个类型里进行？）



### 三、变量定义时的初始化

- 变量必须先赋值再取值。
  - 变量在定义时得到的值是无意义的。如果未赋值就取值，可能发生错误。
- 
- 为了使程序写得简洁，在定义变量时，可以用类似赋值的写法给被定义变量指定初始值，这种描述方式及其效果称为变量的初始化。
  - 程序中通常用简单数值或仅由数值构成的表达式对类型合适的变量进行初始化。

【例3-2】采用定义变量时初始化，改写例3-1程序。

```
#include <iostream>
#include <cmath>
using namespace std;

int main () {
    double a = 3, b = 5, c = 7;
    double s = (a + b + c) / 2.0;
    double area = sqrt(s * (s - a) * (s - b) * (s - c));
    cout << "Area = " << area << endl;
    return 0;
}
```

相同的程序功能完全可能通过不同的语句序列来实现

```
int main () {  
    double a = 3, b = 5, c = 7, s;  
    s = (a + b + c) / 2.0;  
    cout << "Area = " << sqrt(s * (s - a) * (s - b) * (s - c))  
        << endl;  
    return 0;  
}
```

```
int main () {  
    double s = (3 + 5 + 7) / 2.0;  
    cout << "Area = " << sqrt(s * (s - 3) * (s - 5) * (s - 7))  
        << endl;  
    return 0;  
}
```

## 四、赋值运算符的值



- 赋值运算也有值，就是赋给赋值运算符左边的变量的值。
- 赋值表达式的值通常不用。

赋值表达式的值有时用于为多个变量赋值：

**y = (z = (x = 1.0));**

赋值运算符从右向左结合。

上面语句可以简化（也称多重赋值）：

**y = z = x = 1.0;**

- 赋值表达式的值还可以用于流程控制（见后文）

问：下面在变量定义时的初始化语句是否正确？

```
int main() {  
    int i = j = k = 3;  
}
```

答：不正确。

这个语句可以加括号改写为 **int i=(j=(k=3));**

意味着需要事先定义变量 **j** 和 **k**，然后再用于给新定义的变量 **i** 进行初始化。显然前面还并没有定义变量 **j** 和 **k**，所以编译时会出错。

## 五、对求值顺序敏感的表达式

赋值表达式有值，下面是合法语句：

```
x = 2.0;
```

```
y = (x = 3.0) + x;
```

执行后 **y** 的值是什么？

这个问题没有回答，因为第二个语句不正确。

**C/C++**没有规定加法对两个运算对象的求值顺序，所以，“这种表达式的结果没有定义”。

不应该写那种依赖于特殊计算顺序的表达式

## 六、变量赋值与复合赋值运算符

虽然用等于符号作为赋值符号，但赋值与数学中的等于关系是完全不同的两个概念。

**$k = k + 1;$**

取出变量  **$k$**  当时的值加**1**，把得到的结果再赋给变量  **$k$** 。执行效果就是使变量  **$k$**  的值增加了**1**。

在程序中还常常需要下面这类的语句：

**$k = k - 2;$        $k = k * 2;$**

**$k = k / 2;$        $k = k \% 2;$**

取出变量  **$k$**  当前的值，然后进行相应的算术运算，并把结果重新赋给变量  **$k$** 。

语言中专门为此设计了复合赋值运算符:

**+=** 加法赋值      **-=** 减法赋值      **\*=** 乘法赋值  
**/=** 除法赋值      **%=** 模运算赋值

利用这几个运算符可以更简洁地描述一些修改变量操作。例如，上面五条语句可以改写如下：

**k += 1;**                      **k -= 2;**  
**k \*= 2;**                      **k /= 2;**      **k %= 2;**



## 简单计算程序

```
#include <stdio.h>
```

```
/* 如用数学函数，要写#include <math.h> */
```

```
int main () {
```

```
    /* 若干变量定义（及可能的初始化） */
```

```
    /* 若干计算和赋值语句 */
```

```
    /* 若干输出语句 */
```

```
    return 0;
```

```
}
```

这是 **C/C++** 程序的基本形式，后面逐步扩充。

## 七、增量和减量运算符（++、--）

在程序中经常要对变量加 1 或减 1，语言为此提供了专门的增量和减量运算符：**++** **--**

都有前置写法和后置写法：

	前置写法	后置写法
将变量 <b>k</b> 的值增加 <b>1</b>	<b>++k</b>	<b>k++</b>
将变量 <b>k</b> 的值减少 <b>1</b>	<b>--k</b>	<b>k--</b>

**单独**使用增量和减量运算符写成语句时，前置写法和后置写法是等价的。例如：

语句 “**k++;**” 与 “**++k;**” 都使 **k** 的值增加 1，

语句 “**k--;**” 与 “**--k;**” 都使 **k** 的值减少 1。

前置写法与后置写法**作为表达式**求出的值不同：

前置写法 **++k** 求出的是 **k** 加 **1** 之**后**的值；

后置写法 **k++** 的值是 **k** 加 **1** 之**前**的值。

减量操作情况也类似。

**k = 2;**

**m = 2 + ++k;** // **k**变为3，**m**为5（**++k** 的值是加1之后的值）

**n = 3 + k++;** // **k**变为4，**n**为6（**k++** 的值是加1之前的值）

（可以这样通俗地理解：前置写法是先执行增减量运算、后做其它运算；后置写法是先做其它运算、后做增减量运算）

**【例3-3】** 为了让读者对上述的变量操作和相关运算符有更深刻的理解，下面提供一个简单的测试程序（读者也可以自行编写更多语句进行测试）。

```
int main() {  
    cout << "变量相关知识测试" << endl << endl;  
    int m, n; //定义变量  
    cout << "定义后未赋初值: m = " << m << " n = " << n << endl << endl;  
    //未赋初值就取值，错误！运行时所输出的m和n的值无法预料  
    m = 5; //正常赋值  
    n = 5.5; //赋值时有类型转换  
    cout << "已赋初值: m = " << m << "\t n = " << n << endl << endl;  
    m = m + 1;  
    cout << "m 再次赋值之后: m = " << m << endl << endl;  
    cout << "赋值表达式的值: " << (m = 5) << " " << (n = 5.5) << endl;  
    cout << "\n测试复合赋值运算符\n";  
    m -= 4;  
    cout << "m -= 4 之后:\t" << m << endl;  
    n *= 2;  
    cout << "n *= 2 之后:\t" << n << endl;  
    cout << "\n测试增量减量运算符的前置和后置写法\n";  
    m = n = 8;  
    cout << "m = " << m << " n = " << n << endl;  
    cout << "++m : " << ++m << endl;  
    cout << "n++ : " << n++ << endl;  
    cout << "m = " << m << " n = " << n << endl;  
    return 0;  
}
```

## 八、常变量、枚举常量与符号常量

在程序中不同地方，代表同一个常量。

在程序中多个地方统一使用，只需要在定义处进行修改，自动使用新值。

通常用**全大写字母拼写的标识符**作为常量名。

**常变量**通过变量定义语句定义。前面加上修饰符 **const**，还需要直接初始化。

其值只能通过初始化确定，不允许（重新）赋值。

```
const int NUM = 10;
```

```
const double PI = 3.1415927; //圆周率
```

```
const double E = 2.718282; //自然对数的底
```

允许在一个定义语句里定义多个同类型的常变量，允许定义任何类型的常变量。

**枚举常量：**由关键词 **enum** 开始，在紧接着的花括号内写出需要定义常量的名字（标识符）和**整数**常数值。

```
enum { NUM = 10, LEN = 20};
```

只能定义代表整数的常量。

通过预处理命令 “**#define**”定义宏（**macro**）。

```
#define 宏名字 宏常量
```

**宏名字**应是一个标识符，宏常量可以是任意的常量。

其中并没有 “=”号，句末无分号。不是语句，也不是赋值。

由 **#define**开始的行称为**宏定义命令行**。它与 **#include**的用法类似，**通常写在程序的最前面部分**。

```
#define PI 3.14159265
```

以上三种定义符号常量的方法各有特点。根据经验，建议一般采用 **const** 的方式定义常变量，尽量不要使用 **#define**定义宏常量。

**【例3-4】**常变量的定义和使用：考虑定义常变量 **PI**，然后计算给定径 **r = 1.5 m**的圆的周长和面积，并计算给定半径 **r = 2.4 m**的球的体积。

```
int main () {  
    const double PI = 3.14159265;  
    double r;  
    r = 1.5;  
    cout << "r=1.5 的圆的周长: " << 2 * PI * r << " m" << endl;  
    cout << "r=1.5 的圆的面积: " << PI * r * r << " m^2" << endl;  
  
    r = 2.4;  
    cout << "r=2.4 的球体的体积: " << PI * r * r * r * 4/3  
        << " m^3" << endl;  
    return 0;  
}
```

【例3-5】标准库有头文件**limits.h**，其中定义了表示当前系统中**int**类型数据的最小值和最大值的符号常量**INT\_MIN**和**INT\_MAX**，我们希望输出这两个符号常量的值，在超范围加减产生溢出错误的情况，并考虑加减时如何避免产生溢出（本例的目的是复习上一章关于溢出错误的知识）。

```
#include <iostream>
```

```
#include <iomanip>
```

```
#include <limits.h> //提供了各种类型数据的取值范围常量
```

```
using namespace std;
```

```
int main() {
```

```
    cout << "int 类型的最小值和最大值:\t" << INT_MIN << "\t" << INT_MAX << endl;
```

```
    cout << "减1加1溢出得到错误结果:\t" << INT_MIN - 1 << "\t" << INT_MAX + 1 << endl;
```

```
    cout << "转换到双精度数避免了溢出:\t" << setprecision(12) << INT_MIN - 1.0 << "\t" << INT_MAX + 1.0 << endl;
```

```
}
```

int 类型的最小值和最大值:	-2147483648	2147483647
-----------------	-------------	------------

减1加1溢出得到错误结果:	2147483647	-2147483648
---------------	------------	-------------

转换到双精度数避免了溢出:	-2147483649	2147483648
---------------	-------------	------------

整数溢出时，实际上是内部错误地使符号位参与了加减法计算<sub>43</sub>



## 第 3 章 变量和控制结构

**3.1** 语句、复合结构和顺序程序

**3.2** 变量——概念、定义和使用

**3.3** 数据输入

**3.4** 关系表达式与逻辑表达式

**3.5** 语句与控制结构

**3.6** 条件语句

**3.7** 循环语句

**3.8** 程序动态除错方法（一）

## 3.3 数据输入

在**C++** 程序中，基本输入操作通过 **cin** 实现。

**cin**是**C++** 标准库定义的输入流对象，通常与右向双箭头“>>”（“提取运算符”）配合使用，用于从输入流中获取数据并赋给指定的变量。

程序中使用**cin**完成输入的（输入语句）一般描述形式是：

**cin >> 变量名1 >> 变量名2 >> ... >> 变量名n;**


程序运行中遇到这种语句时，程序将暂停在这里，等待用户从标准输入设备（通常是键盘）输入所需项数的数据。

如果语句要求多项数据，在实际输入时，数据之间可以用空格、制表符或回车符分隔。输入完毕所需的所有数据并按回车键之后，**cin** 就会从输入流中获取这些数据，并依次将它们送给语句中描述的变量。

【例3-6】请写出一个程序的主函数，它要求输入一个三角形的三条边长**a**、**b**、**c**，然后先用公式  $s = (a+b+c)/2$  计算出半周长，再用公式  $\sqrt{s(s-a)(s-b)(s-c)}$  算出三角形的面积。

```
int main () {  
    double a, b, c, s;  
    cout << "Please input a, b, c: ";  
    cin >> a >> b >> c;  
    s = (a + b + c) / 2.0;  
    cout << "Area = " << sqrt(s * (s - a) * (s - b) * (s - c))  
        << endl;  
    return 0;  
}
```

要养成一种良好习惯：在每次等待输入之前都输出一句操作提示！



细节： .....

cin 会完成输入数据的自动类型转换  
三角形任意两条边长之和大于第三条边长

### \*3.3.2 格式输入函数scanf

格式输入函数 **scanf** 的功能与 **printf** 对应。

**scanf** 从标准输入读数据，根据格式描述将实际输入转换到指定类型，转换结果赋给指定变量：

**scanf(格式描述串, &变量名, ...)**

**格式描述串**与**printf**的**类似**，其中的转换描述（以**%**开头）说明输入形式和转换方式。

其他参数（个数应与格式串中转换描述一致）指明接受输入的程序变量。形式是在**变量名**前加 **&** 符号。

注意：必须写 **&** 符号，不写将引起严重问题。💥

简单示例：

```
#include <stdio.h>
int main() {
    int n;
    printf("Plsease input a number: ");
    scanf("%d", &n);
    printf("%d %d\n", n, n * n);
    return 0;
}
```

程序执行后输出提示串：

**Please input a number:**

等待人的输入。得到输入数据后输出并结束。程序的行为依赖于当时的输入（与前面程序不同）

## 常用的 **scanf** 转换描述:

转换描述	参数变量类型	所要求的实际输入
<b>%d</b>	<b>int</b>	十进制整数数字序列
<b>%ld</b>	<b>long</b>	同上
<b>%f</b>	<b>float</b>	十进制数，可以有小数点及指数部分
<b>%lf</b>	<b>double</b>	同上
<b>%Lf</b>	<b>long double</b>	同上

注意实数类型的转换描述与 **printf** 的**差异**。

例：设有变量定义：

```
int n; double x; float y;
```

可以写语句：

```
scanf("%d %lf %f", &n, &x, &y);
```

读数值时， **scanf** 格式串里的转换描述之间的空格并不必要。上面语句写成下面形式，效果一样：

**scanf("%d%lf%f", &n, &x, &y);**

如果这里的转换描述之间没字符或只有空格，输入的数据之间也只能有空白字符，不能有其他字符。

格式串里一般**不写转换描述之外的东西**。如果写

**"%d, %lf, %f"**

就是要求用逗号分隔输入数据，**若输入时不注意就会导致数据不能正常读入**。建议**不要**这样写。

**scanf** 格式串的细节在第八章有详细介绍。

# 类型安全性

- 在使用 **printf** 进行输出时，如果格式描述串里的转换描述与实际要输出的值的类型不匹配，产生的输出可能不是用户所需。但转换方式错误不会影响程序本身的行为，不会影响程序随后的执行过程。
- （前面已讲）使用 **scanf** 输入时，必须写 **&** 符号，不写将引起严重问题。
- 使用 **scanf** 输入时，如果格式描述串里的转换描述与实际要输入的值的类型不匹配，会产生严重的无法预料的后果。
- 所以，在写使用**scanf** 进行输入的程序时，必须特别关注转换描述和接受输入的变量在类型方面的一致性。必须时时注意，认真检查。





## 第 3 章 变量和控制结构

**3.1** 语句、复合结构和顺序程序

**3.2** 变量——概念、定义和使用

**3.3** 数据输入

**3.4** 关系表达式与逻辑表达式

**3.5** 语句与控制结构

**3.6** 条件语句

**3.7** 循环语句

**3.8** 程序动态除错方法（一）

## 3.4 关系表达式/逻辑表达式/条件表达式

已有机制的编程能力还很弱。

在解决实际问题时，经常需要检查实际数据，根据不同情况采取不同的处理方式。

为此就需要对变量或数据进行比较，判断某个条件是否成立。

描述比较操作、完成判断，就需要进行关系运算和逻辑运算。

### 3.4.1 关系运算符和关系表达式

关系运算符确定数据间是否存在某种关系。

关系运算符共 6 个：

>   >=   <=   <   大于/大于等于/小于等于/小于  
==   !=   等于/不等于

对各种数值类型都可以使用这些关系运算符，写出关系表达式。通常把关系表达式用于程序控制。

如果被比较对象的类型不同，按算术运算规则转换后再做判断。关系的成立与否（真/假）：

3.24 <= 2.98    5 != 3+4    5.0 < 7-4

i>10   y != x + 1    (int)2.0 == 2   fabs(x-2.0)<1e-6

## 逻辑值

关系式求值的结果是成立/不成立。



描述逻辑判断/逻辑性质：

- 关系成立，所描述的关系“真”，逻辑值“真”；
- 不成立时该关系“假”，逻辑值“假”。

程序用逻辑判断/逻辑值控制计算进程。

在 ANSI C 中，关系运算的结果为 **int** 类型，成立/不成立时的值是 **1/0**。

**C99**和**C++** 中，提供了逻辑值类型**bool**，定义了两个逻辑值**true**和**false**分别表示“真”和“假”。

**bool**类型变量只能取 **true / false**（只能取**1 / 0**）

对关系表达式求值，将得到**bool**类型的值：当相应关系成立时得到**true**；关系不成立时得到**false**。

关系表达式“**3.24 <= 2.98**”的值就是**false**，而“**5 != 3 + 1**”的值是**true**。

## 关系表达式的求值:

关系表达式	人脑逻辑判断	表达式的值
$2 > 1$	真(True)	1
$3.2 \leq 2.9$	假(False)	0
$3.24 == 2.98$	假(False)	0
$5 != 3 + 1$	真(True)	1
$m > 10$	真(True)	1
$n < m$	假(False)	0

逻辑值

假设已有 `int m=20, n=100`

C/C++里的“逻辑值”比人类生活中的“逻辑判断”有扩展。

C/C++中所有基本类型的值都可当作逻辑值使用：

- 值等于 0 表示逻辑值 “假 (0)”
- 非 0 值 都作为逻辑值 “真(1)”



数值或表达式	人脑逻辑判断	逻辑值
0		0
2		1
2.0		1
5-3		1
0.2*10		1
'a'		1

学习难点解析：

算术表达式和赋值表达式有返回值，都可以当作逻辑值来用。

**int m=20;**

表达式	返回值	逻辑值
<b>m&gt;0</b>	<b>1</b>	<b>1</b>
<b>m==20</b>	<b>1</b>	<b>1</b>
<b>m-20</b>	<b>0</b>	<b>0</b>
<b>m+1</b>	<b>21</b>	<b>1</b>
<b>m=10.1</b>	<b>10</b>	<b>1</b>
<b>m=0</b>	<b>0</b>	<b>0</b>
<b>(m==1)&gt;=1</b>	<b>0</b>	<b>0</b>
<b>(m=1)&gt;=1</b>	<b>1</b>	<b>1</b>



【例3-7】使用 `cout <<` 方法，计算一些关系表达式的值并输出到屏幕。

```
int main() {  
    cout << "测试关系表达式的值\n\n";  
    cout << "2 > 1 \t" << (2 > 1) << endl;  
    cout << "1 >= 0 \t" << (1 >= 0) << endl;  
    cout << "3.24 <= 2.98 \t" << (3.24 <= 2.98) << endl;  
    cout << "6 <= 12 / 2 \t" << ( 6 <= 12 / 2) << endl;  
    cout << "5 == 3 + 1 \t" << (5 == 3 + 1 ) << endl;  
    cout << "10 != 2 * 5 \t" << (10 != 2*5) << endl << endl;  
  
    int k = 10;  
    cout << "k = " << k << endl;  
    cout << "k > 100 \t" << (k > 100) << endl;  
    cout << "k >= 10 \t" << (k >= 10) << endl;  
    cout << "k == 10 \t" << (k == 10) << endl;  
  
    bool logic = true; //定义 bool 类型的变量  
    logic = (k > 10);  
    cout << "logic= " << logic << endl;  
    return 0;  
}
```

同学们应该自己动脑筋，  
写一些复杂的关系表达式，  
求值看看结果。

## 3.4.2 逻辑运算符与逻辑表达式

三个逻辑运算符：

**&&**、**||** 和 **!**：“并且(与)”、“或者(或)”和“否定(非)”

**&&** 和 **||** 是二元，**!** 是一元。

表达式1 **&&** 表达式2

“与”：两表达式的值都非 0，结果为 1，否则为 0

表达式1 **||** 表达式2

“或”：两表达式的值只要有一个非0，结果为1。  
否则为0。

**!** 表达式

“非”：表达式看作逻辑值，以该值的否定为结果

利用逻辑运算符可以描述复杂的关系

$m > 10 \ \&\& \ n < 0$        $m > 10 \ || \ n < 0$

$k \geq 3 \ \&\& \ k \leq 5$        $k < 3 \ || \ k > 5$

$('A' \leq ch \ \&\& \ ch \leq 'Z') \ || \ ('a' \leq ch \ \&\& \ ch \leq 'z')$

**&& 和 || 的计算方式：**先算左边运算对象，如果可确定结果，右边运算对象将不计算。

例：       $x \neq 0.0 \ \&\& \ y/x > 1.0$

求值时不会出现除 0 问题。

例： 数学公式 “  $x < 3$  或  $x > 5$  ” 写作  $x < 3 \parallel x > 5$

数学公式 “  $3 \leq x \leq 5$  ” 写作  $x \geq 3 \&\& x \leq 5$

思考： 上式为什么不能直接写成  $3 \leq x \leq 5$  ？

答：

如果在源程序写成  $3 \leq x \leq 5$ ， 会被解释为

$(3 \leq x) \leq 5$ ， 其值为：

$3 \leq x$  的值为 0 或 1，  $0 \leq 5$  和  $1 \leq 5$  的值都是 1。

逻辑运算符 **!** 将把运算对象表达式的值看作逻辑值  
(如果原来不是逻辑值, 就会自动转换为逻辑值),  
以该值的否定作为结果:

**!(m > 10)**

**!(m > 10 && n < 0)**

**!(k < 3 || k > 5)**

**!15**

**!k**

逻辑表达式计算结果都是整型的 **0** 或 **1**。

否定的优先级同其他一元运算符；

**&&** 优先级高于 **||**，低于关系运算符。

例：根据运算符优先级关系，逻辑表达式：

**$((x+3) > (y+z)) \&\& (y < 10) || (y > 12)$**

写为下面形式意义不变（但是比较难懂）。

**$x + 3 > y + z \&\& y < 10 || y > 12$**

所以，为了便于阅读，适当加括号方便读懂。

关于逻辑运算符的额外说明：

- (1) 逻辑运算符 **&&**、**||** 和 **!** 分别有别名（规范运算符名之外的名称）**and**、**or** 和 **not**。使用这些别名可以提高程序的可读性。例如 “**(ch>='A' && ch<='Z') || (ch>='a' && ch<='z')**” 可以写成 “**(ch>='A' and ch<='Z') or (ch>='a' and ch<='z')**”
- (2) 在书写逻辑运算符 **&&** 和 **||** 时，**不要误写成单个字符的 **&** 或 **|****。写错时编译器一般不会报错。原因是，语言里还有四个字位运算符，字位“否定” **~**，字位“或” **|**，字位“与” **&**，字位“异或” **^**。它们有特殊的用途，初学者不会用到它们的功能，但要注意不写错。

**【例3-8】** 使用 `cout <<` 方法，计算一些关系表达式和逻辑表达式的值并输出到屏幕。

```
int main() {  
    int m = 15, n = 3;  
    cout << "m= " << m << " n= " << n << endl;  
    cout << "(m > 10) \t" << (m > 10) << endl;  
    cout << "!(m > 10) \t" << !(m > 10) << endl;  
    cout << "(n < 0) \t" << (n < 0) << endl;  
    cout << "!(n < 0) \t" << !(n < 0) << endl << endl;  
  
    cout << "(m > 10 && n < 0) \t" << (m > 10 && n < 0) << endl;  
    cout << "!(m > 10 && n < 0) \t" << !(m > 10 && n < 0) << endl << endl;  
    int k = 4;  
    cout << "k= " << k << endl;  
    cout << "(k >= 3 && k <= 5) \t" << (k >= 3 && k <= 5) << endl;  
    cout << "(k < 3 || k > 5) \t" << (k < 3 || k > 5) << endl << endl;  
    k = -10;  
    cout << "k= " << k << endl;  
    cout << "(k >= 3 && k <= 5) \t" << (k >= 3 && k <= 5) << endl;  
    cout << "(k < 3 || k > 5) \t" << (k < 3 || k > 5) << endl;  
  
    cout << "!k \t" << (!k) << endl;  
    cout << "!(k + 10) \t" << !(k + 10) << endl;  
    cout << "!(k == 0) \t" << !(k == 0) << endl;  
    return 0;  
}
```



### 3.4.3 条件表达式

条件运算符“?:”。条件表达式的语法形式:

表达式1 ? 表达式2 : 表达式3

语义（计算方式特殊）:

- 先算表达式1（关系表达式或逻辑表达式或算术表达式）;

> >= < <= == !=      && || !      + - \* / % 函数

以其值作为逻辑值。

- 条件成立时算表达式2，以其结果作为条件表达式的值
- 条件不成立算表达式3，以其值作为条件表达式的值

注意：条件成立时不算表达式3；不成立时不算表达式2

**【例3-9】** 使用条件表达式计算某变量 **x** 的绝对值和正负符号。

计算变量 **x** 的绝对值可以用如下条件表达式：

**x >= 0 ? x : -x**

符号函数的数学定义是：

$$\text{sign}(x) = \begin{cases} 1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0 \end{cases}$$

在变量 **x** 的值大于、等于或小于 **0** 时，结果分别为**1**、**0** 和 **-1**，利用嵌套的条件表达式写出如下：

**x > 0 ? 1 : (x == 0 ? 0 : -1)**

写在一个**main**函数里，查看它们的计算结果：

```
int main() {  
    double x;  
    cout << "please input x: ";  
    cin >> x;  
    cout << "abs of x: " << (x >= 0 ? x : -x) << endl;  
    cout << "sign of x: " << (x > 0 ? 1 : (x == 0 ? 0 : -1))  
        << endl;  
    return 0;  
}
```